VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Program pro práci s třídícími sítěmi

Program for Manipulation of Sorting Networks

2017 Josef Pohrom

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Josef Pohrom**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Program pro práci s třídícími sítěmi
Program for Manipulation of Sorting Networks

Jazyk vypracování: čeština

Zásady pro vypracování:

Třídící sítě (sorting networks) představují jednoduchý matematický model pro popis paralelních třídících algoritmů. Třídící síť si můžeme představit jako určitý druh obvodu, kde "vodiče" přenášejí tříděné hodnoty. Jedinými prvky obvodu jsou porovnávací členy, které umí pouze porovnat dvě hodnoty a poslat na jeden svůj výstup menší z těchto hodnot a na druhý větší z těchto hodnot.
Když na vstupy třídící sítě přivedeme hodnoty, které chceme setřídit, na výstupech dostaneme tyto hodnoty setříděné.

Cílem této bakalářské práce je vytvořit program pro interaktivní práci s třídicími sítěmi. Program by měl poskytovat grafické uživatelské rozhraní pro vytváření, editování a testování těchto sítí. Kromě ručního interaktivního vytváření sítí by měl program umožňovat programové generování sítí, tj. popsat konstrukci příslušné sítě pomocí programu v nějakém obecném programovacím jazyce.

1. Nastuduje problematiku třídících sítí.
2. Navrhněte vhodné uživatelské rozhraní pro práci s třídícími sítěmi.
3. Implementujte program pro interaktivní práci s třídícími sítěmi. Kromě interaktivní práce by měl program umožňovat i programové vytváření třídících sítí.
4. Implementujte v programu funkce pro testování korektnosti vytvořených třídících sítí.

Seznam doporučené odborné literatury:

[1] A. Gibbons, W. Rytter, Efficient Parallel Algorithms, Cambridge University Press, 1990.
[2] Chapter 27 - Sorting Networks from T. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms (1st ed.), MIT Press, 1990.

Další literatura podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Ing. Zdeněk Sawa, Ph.D.**

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 28. dubna 2017 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 28. dubna 2017 .....

Na tomto místě bych chtěl poděkovat doc. Ing. Zdeňku Sawovi, Ph.D za příkladné vedení této bakalářské práce a cenné rady, odborné konzultace a velice vstřícný přístup při tvorbě této bakalářské práce.

Abstrakt

Bakalářská práce se zabývá vytvořením programu pro práci s třídícími sítěmi. V úvodní části je, po seznámení se se základními pojmy, provedena analýza celého zadání a následně popsáno navržení programu. V další části je vysvětlena implementace některých části programu a je zde uveden stručný uživatelsky návod, jak s programem pracovat. Cílem bakalářské práce je vytvořit program pro interaktivní práci s třídícími sítěmi, který by měl poskytnout grafické uživatelské rozhraní, pro vytváření, editování a testování těchto sítí.

Klíčová slova: třídící sítě, algoritmizace, paralelní algoritmy

Abstract

The bachelor thesis deals with the creation of a program for working with sorting networks. In the introductory part, after the familiarization with the basic concepts, the entire assignment is analyzed and the program design is described. The next section explains the implementation of some parts of the program and gives a brief user guide on how to work with the program. The aim of the bachelor thesis is to create a program for interactive work with sorting networks, which should provide a graphical user interface for creation, editing and testing of these networks.

Key Words: Sorting networks, algorithms, parallel algorithms

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Při zkoumání paralelních třídících obvodů se zavádí pojem třídící síť. Je to jednoduchý matematický model pro popsání paralelních třídících algoritmů. Hlavním prvkem těchto obvodů jsou komparátory. Každý komparátor dostane na vstupu dvě hodnoty a ty poté na svém výstupu vrací setříděné. Na výstupu celého obvodu by pak měly být tyto hodnoty korektně setříděny.

Mým úkolem v této bakalářské práci bylo navrhnout a následně naprogramovat aplikaci, která umožní v grafickém prostředí uživateli navrhovat a následně testovat korektnost jim navržené třídící sítě. Program slouží k ulehčení návrhu paralelních třídících algoritmů (třídících sítí), protože umožňuje zobrazení animace, která znázorňuje, jak se hodnoty v třídící síti postupně třídí a to usnadní odhalování konkrétních nekorektností vytvořených v obvodu. Při testování si lze vybrat, jak obsáhlý test bude (kolik náhodných posloupností program nechá setřídit), a tím zvýšit šanci na odhalení chyb v třídící sítí.

Práce je strukturována do čtyř hlavních kapitol, které vysvětlují problematiku této práce a jednotlivé kroky vývoje.

Kapitola 2 je zaměřena na vysvětlení problematiky třídících sítí, seznámení s pojmy, které se při tvorbě třídících sítí vyskytují. Poté jsou uvedeny konkrétní příklady již sestrojených algoritmů a jejich stručné vysvětlení, jak fungují.

V Kapitole 3 byla provedena celková analýza zadání bakalářské práce a následného výsledku po vypracování. Dále byly zohledněny použité technologie pro vývoj této aplikace. Následně je poukázáno na celkový návrh aplikace, jaké třídy používá a jaký mají tyto třídy úkol v programu.

V Kapitole 4 je podrobně popsána implementace aplikace. Je zde vysvětleno, jak jsou implementovány algoritmy pro načítání třídících sítí, testování jejich korektnosti a v neposlední řadě jejich simulace.

Kapitola 5 poté slouží jako stručný uživatelský návod, který seznámí uživatele s požadavky pro instalaci a následně jak pracovat s programem.

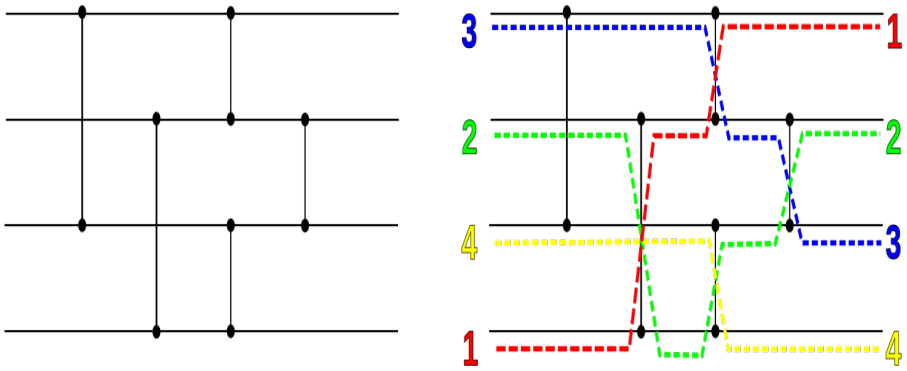
2 Třídící sítě

V této kapitole popisuji, jak vlastně třídící sítě fungují a vysvětluji základní pojmy, které jsou v třídících sítích využívány. Na závěr této kapitoly ukáži příklady některých algoritmů.

2.1 Co jsou třídící sítě

Třídící sítě jsou tvořeny komparátory a vodiči. Každý vodič nese jednu hodnotu zleva doprava. Všechny hodnoty sítí procházejí najednou (paralelně). Komparátor je připojen právě ke dvěma vodičům, které nesou hodnoty a slouží k porovnání hodnot. Když hodnoty dorazí ke komparátoru, tak komparátor tyto dvě hodnoty porovná a na jeden výstup pošle větší hodnotu a na druhý výstup hodnotu menší, to znamená, že po průchodu hodnot komparátorem jsou tyto hodnoty setříděny. Výsledkem všech těchto porovnání by měla být na výstupu setříděná posloupnost hodnot. Síť, která tuto podmínku splní se může nazývat třídící sítí.[1]

Na Obrázku 1 lze vidět jednoduchý příklad třídící sítě obsahující 4 vodiče a 5 komparátorů. Z obrázku lze lehce vypozorovat, proč tato třídící síť setřídí korektně všechny vstupní hodnoty. První čtyři komparátory přesunou větší hodnotu na spodek sítě a menší hodnoty přemístí na vrch sítě. Poslední pátý komparátor již jen dotřídí zbylé dvě prostřední hodnoty. Na výstupu sítě lze vidět vzestupně setříděnou posloupnost hodnot. [1]



Obrázek 1: Ukázka jednoduché třídící sítě
[1]

Hloubka třídící sítě

Každá třídící síť je sestavena z n komparátorů, které třídí přivedené hodnoty. Hloubka třídící sítě určuje největší počet komparátorů, který je připojen k jednomu vodiči. Na Obrázku 1 lze vidět, že na druhém vodiči jsou připojeny tři komparátory a na ostatních vodičích pouze dva. To znamená, že hloubka této třídící sítě je tedy 3.[1]

Seznam pojmů

Komparátor - propojuje právě dva vodiče a porovnává hodnoty
Vodič - přenáší hodnotu zleva doprava
Hloubka - určuje největší počet komparátorů připojených k jednomu vodiči

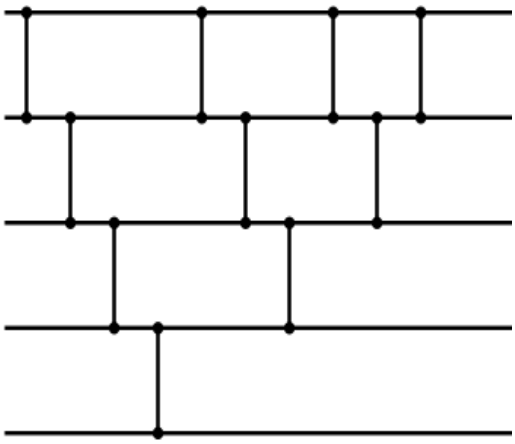
2.2 Příklady algoritmů

Za pomoci třídících sítí lze sestavit mnoho různých algoritmů. Jako prvním jednoduchým algoritmem je Bubble sort (bublinkové třídění).

(a) **Bubble Sort**

Tento algoritmus porovnává vždy sousedící prvky, pokud je horní prvek větší než spodní, tak se tyto dva prvky vymění. Stejným principem algoritmus pokračuje až k poslední hodnotě v třídící síti. Na konci každé iterace se dostane největší hodnota na dno sítě. Tento algoritmus má složitost $\Theta(n^2)$.[2]

Na Obrázku 2 jde vidět, jak by tento algoritmus vypadal zkonstruován jako třídící síť.

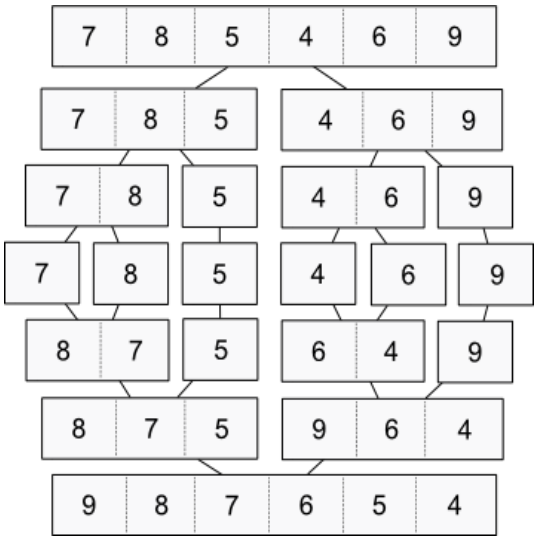


Obrázek 2: Ukázka Bubble sortu vytvořeného jako třídící síť

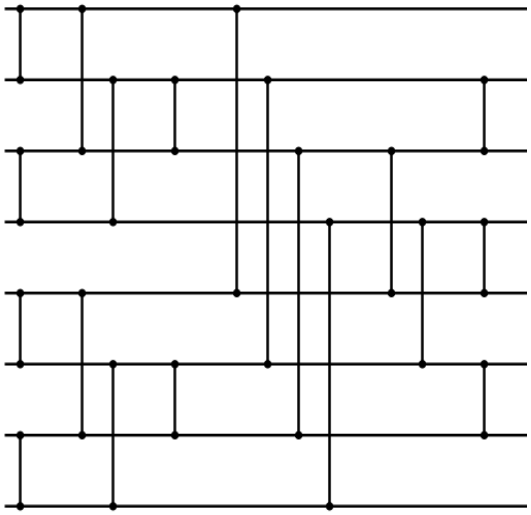
(b) **Merge sort**

Algoritmus začíná „rozdělováním“ všech hodnot vždy na dvě stejné části (pokud je na vstupu 6 hodnot, tak při prvním rozdělení je rozdělí na polovinu tzn. 3 a 3) a rekurzivně pokračuje, dokud všechny hodnoty nerozdělí do jednotkové velikosti (hodnoty jsou triviálně setříděné). Když se algoritmus dostane do této úrovně začíná zpětně tyto hodnoty slévat do setříděné posloupnosti. Tento algoritmus má složitost $\Theta(n \log n)$. [3]

Ukázku algoritmu lze vidět na Obrázku 3a a jak by tento algoritmus vypadal jako třídící síť lze vidět na Obrázku 3b.



(a) Ukázka průběhu Merge sortu [3]



(b) Ukázka Merge sortu vytvořeného jako třídící síť

Obrázek 3: Merge sort

3 Analýza a návrh

V této kapitole popisuji analýzu všech požadavků, které jsou v zadání bakalářské práce, jak jsem postupoval při analýze celého řešení, proč jsem zvolil tato řešení a jaké technologie jsem použil. Dále popisuji, jak je celá aplikace navrhnuta a co dané třídy dělají a jaké mají vlastnosti.

3.1 Analýza

V prvním kroku bylo nutné zanalyzovat celé zadání, co vše program musí umět a jaké požadavky splňovat.

Výsledek

Výsledkem bude desktopová aplikace, která umožní uživateli vytvářet třídící sítě v interaktivním prostředí, kde tyto třídící sítě může kreslit nebo za pomoci jednoduchého skriptovacího jazyka si tuto síť „naprogramovat." U všech třídících sítí bude pak uživatel moci otestovat, zda je vytvořil korektně.

Použité technologie

Ze zadání je jasné, že program bude obsahovat grafickou část (bude vykreslovat vodiče a komparátory). Musel jsem tedy vhodně zvolit technologie pro vytvoření teto aplikace. Celá aplikace je vytvořena na windows frameworku .NET v programovacím jazyce C#. Okenní aplikace pak fungují na technologii WPF (Windows Presentation Foundation), což je knihovna pro tvorbu grafického prostředí (GUI) desktopových aplikací.

Funkcionalita programu

Program obsahuje grafické prostředí, ve kterém je možné vytvářet a navrhovat třídící sítě, uživatel bude moci přidávat komparátory na vodiče, odebírat je, popřípadě je přesouvat dle svého uvážení. Takto vytvořené sítě poté bude moci otestovat, jestli jsou korektně navrhnuty (setřídí vždy všechny hodnoty do vzestupného pořadí). Program umožňuje tento test i zobrazit jako animaci, kde znázorní, jak by probíhalo třídění ve fyzickém obvodu (paralélní průběh komparací). Program musí umožňovat načítání třídících sítí pomocí textového skriptu. Tento jednoduchý skriptovací jazyk je vytvořený přímo pro tuto aplikaci a umožní uživateli vytvořit si třídící síť jednoduchým programováním. Vytváření pomocí skriptu je více flexibilní, než vytváření v grafickém editoru. Všechny sítě mohou být uloženy pro jejich pozdější editaci, testování nebo simulaci průběhu třídění.

3.2 Návrh

Zde popíši, jak je celá aplikace navrhnuta, její vnitřní struktura, tak jak funguje z vnějšku. Celou aplikaci jsem musel rozdělit na dvě části, logickou a grafickou. Všechny třídy v grafické části jsou závislé na třídách z logické části ne naopak, protože to usnadní práci s vykreslováním.

Logická část

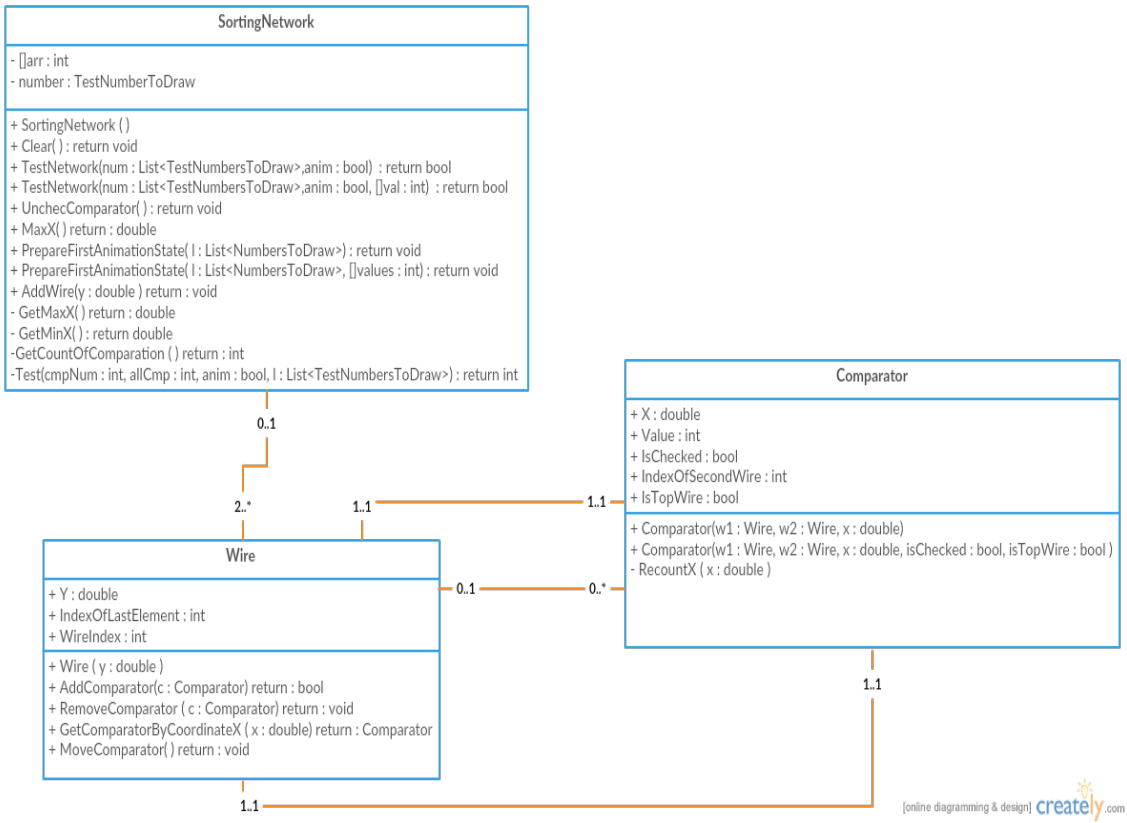
Obsahuje tři hlavní objekty ***SortingNetwork***, ***Wire***, ***Comparator***. Tyto třídy jsou na sobě závislé a tvoří celkovou strukturu třídící sítě, kterou tvoří uživatel.

(a) **SortingNetwork** - třída (class), která udržuje veškeré reference na jí podřízené objekty (Wire) v kolekci, protože vztah mezi touto třídou a třídou typu Wire je 1:N. Obsahuje také metodu pro otestování její korektnosti, která vrací boolovské hodnoty pravda (true), když třídící síť setřídí hodnoty korektně (vzestupné pořadí) nebo nepravda (false) pokud třídící síť není korektně setříděná. Tento objekt má v programu vždy pouze jednu instanci, protože program neumožňuje pracovat najednou s více třídícími sítěmi.

(b) **Wire** - třída (class), která si udržuje seznam o všech komparátorech (Comparator), které jsou k ní připojeny, všechny tyto objekty se udržují v setříděném pořadí podle souřadnice X a opět vztah mezi touto třídou a třídou typu Comparator je 1:N. Dále udržuje vlastnost (*IndexOfLastElement*), který komparátor byl naposledy navštívený, která je nezbytná, pro korektní průchod strukturou (třídící sítí).

(c) **Comparator** - třída (class), která si udržuje reference na vodiče (Wire), ke kterým je připojena. Tato třída si udržuje informaci, kde na vodiči se nachází (souřadnice X). Při průchodu strukturou (třídící sítí) si tento objekt udržuje příznak, jestli už byl navštíven *IsChecked*, aby se zamezilo jeho opětovnému procházení. Velmi důležitou vlastností je *Value*, kde si objekt udržuje jaká hodnota se bude třídit, tato hodnota se nastavuje až při spuštění testu třídící sítě. Udržuje si pouze jednu hodnotu k třídění, protože jeden komparátor je tvořen dvěma instancemi této třídy, které na sebe znají referenci.

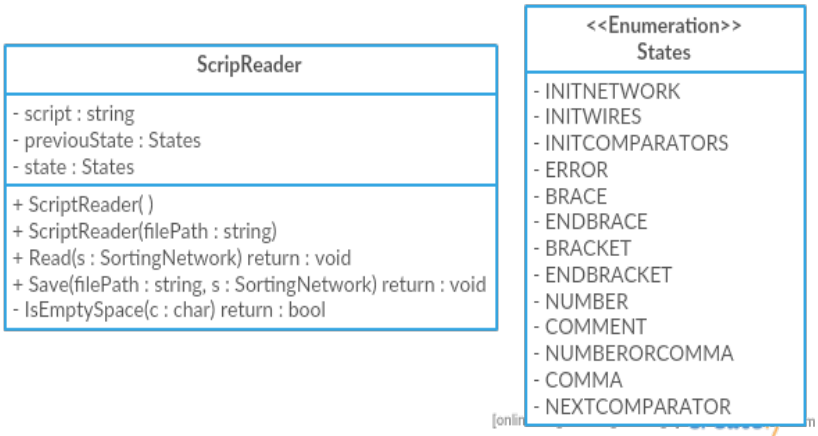
Na Obrázku 4 lze vidět diagram tříd (class diagram), jak je navrhnuta vnitřní logická struktura třídící sítě programu, jaké používá metody a jaké má vlastnosti.



Obrázek 4: Třídní diagram (class diagram) logické struktury sítě

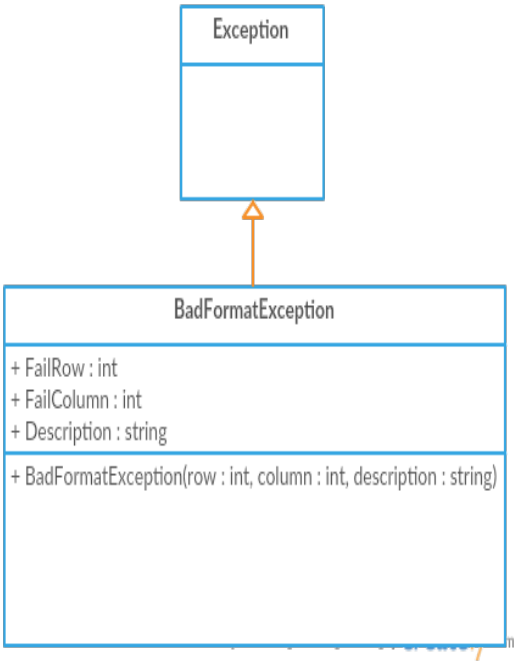
Další dvě třídy z logické části programu jsou ***ScriptReader*** a ***BadFormatException***, které obstarávají načítání uživatelem vytvořenou třídící síť za pomoci skriptu.

(d) **ScriptReader** - třída (class), která zajišťuje načtení třídící sítě z uživatelem napsaného skriptu. Obsahuje vlastní výčtový typ (enum), který obsahuje veškeré stavy, které mohou nastat při načítání třídící sítě z uživatelem vytvořeného skriptu. Proměnná *script* má načteny celý uživatelem vytvořený skript a umožňuje s ním dále pracovat v objektu. Tato třída má dále dvě důležité metody *Read*, která načítá třídící síť z uživatelem napsaného skriptu a metodu *Save*, která ukládá uživatelem vytvořenou třídící síť v grafickém prostředí, do textového souboru, ze kterého lze opět tuto síť načíst. Na Obrázku 5 lze vidět seznam všech vlastností a metod, které třída obsahuje.



Obrázek 5: Seznam vlastností a metod třídy *ScriptReader*

(d) **BadFormatException** - třída, která některé vlastnosti dědí z třídy Exception (tato třída je obsažena v .NET frameworku). Tato třída slouží k vytváření výjimek za běhu programu při načítání uživatelem napsaného skriptu. Tato třída obsahuje tří vlastnosti *FailRow* číslo řádku, na kterém byla nalezena chyba v souboru chyba, *FailColumn* pozice kurzoru na řádku, kde byla nalezena chyba při načítání a *Description* slovní popis chyby, která nastala. Na Obrázku 6 lze vidět seznam atributů této výjimky.

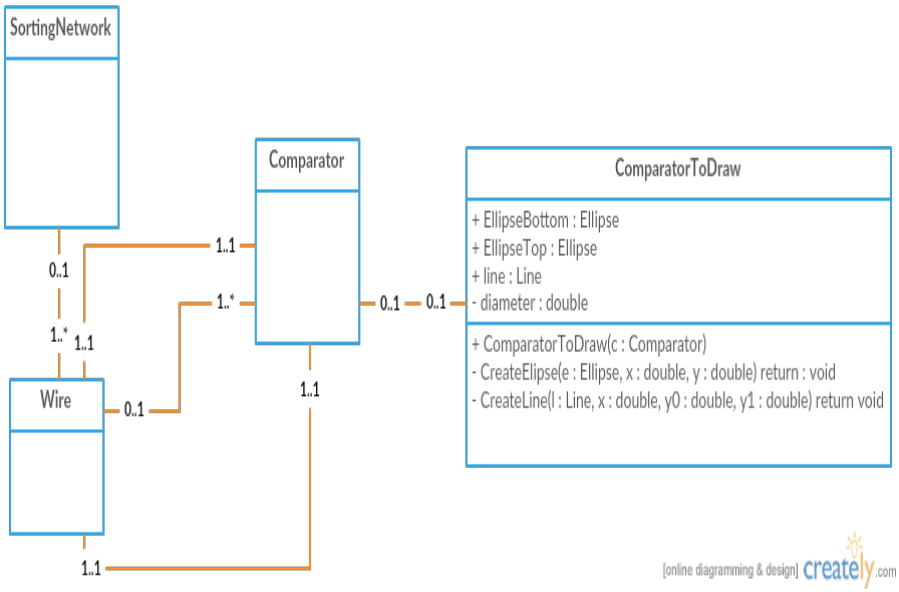


Obrázek 6: Seznam vlastností a metod třídy *BadFormatException*

Grafická část

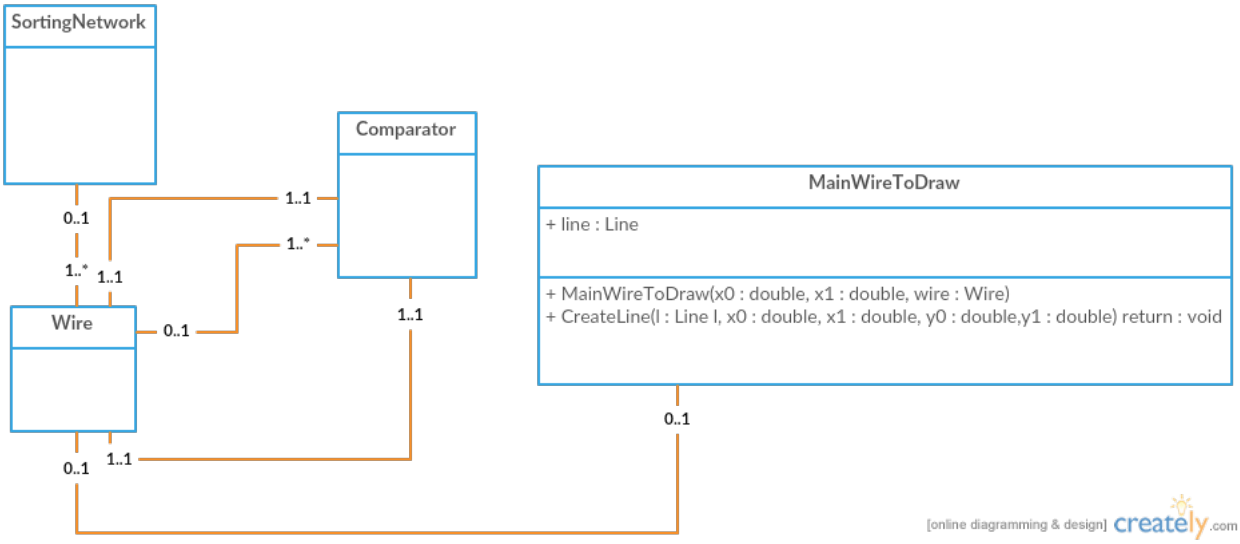
Třídy v této části slouží k grafickému zobrazení komponent na pracovní ploše. Celkem jsou tyto třídy čtyři. ***ComparatorToDraw***, ***MainWireToDraw***, ***TestAnimation*** a ***TestNumberToDraw***.

(a) **ComparatorToDraw** - třída, která obsahuje reference na objekty, které se budou vykreslovat na pracovní plochu. Tyto objekty jsou již obsaženy .NET frameworku *Line* (čára komparátoru) a *Ellipse* (bod uchycení komparátoru). Referenci na komparátor (třída Comparator) z logické struktury je zde taky udržována pro snadnější interaktivní procházení struktury. Každá instance této třídy tedy obsahuje dvě elipsy (horní a spodní) a čáru, která tyto elipsy spojuje. Na Obrázku 7 je zobrazeno, jak je tato třída zakomponována do logické struktury.



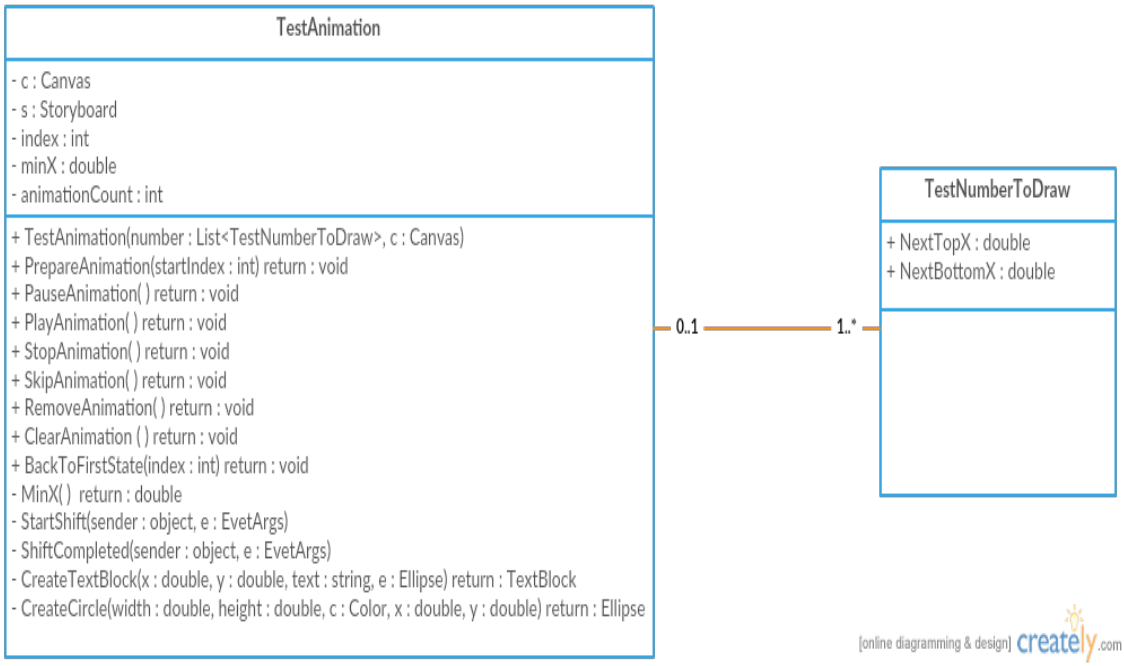
Obrázek 7: Třídní diagram třídy *ComparatorToDraw*

(b) **MainWireToDraw** - třída obsahuje referenci na objekt *Line*, která představuje vodič (*Wire*) čili horizontální čáru, po které probíhají hodnoty k porovnávání a jsou k ní připojeny komparátory (Comparator). Dále zná referenci na vodič (třída Wire) z logické struktury. Instance těchto tříd jsou vytvořeny při založení nového projektu nebo při načtení uživatelského skriptu pro vytvoření třídící sítě. Třídní diagram této třídy lze vidět na Obrázku 8.



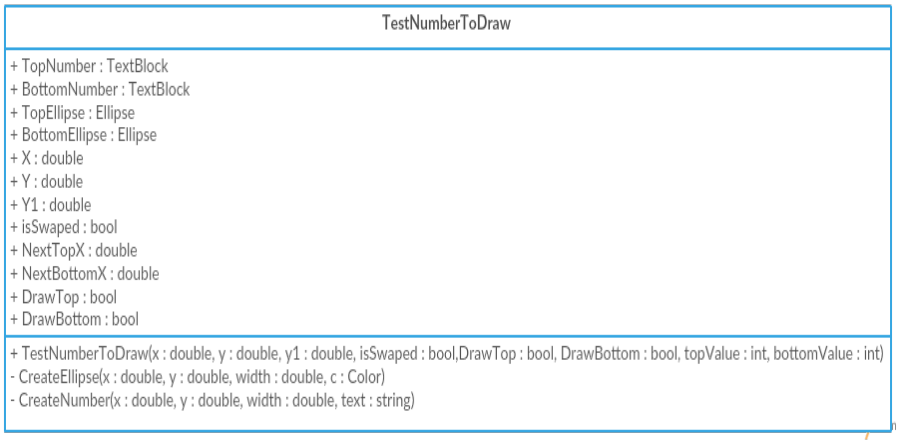
Obrázek 8: Třídní diagram třídy *MainWireToDraw*

(c) **TestAnimation** - třída, která zpracovává graficky animaci průběhu hodnot uživatelem vytvořené třídící sítě. Udržuje referenci na kolekci objektů *TestNumberToDraw*, které následně animuje. K animacím využívá třídu Storyboard, která je obsažena v .NET frameworku. Obsahuje také metody pro ovládání animace (Play, Pause, Stop, Skip). Na Obrázku 9 je vidět třídní diagram a seznam funkcí a vlastností této třídy.



Obrázek 9: Třídní diagram třídy *TestAnimation*

(d) **TestNumberToDraw** - třída související s předešlou třídou (*TestAnimation*), slouží k grafickému vykreslení hodnot na vodičích (Wire). Udržuje si informaci o objektech, které se budou vykreslovat na pracovní plochu (čísla a elipsy) a dále má informaci o vzdálenosti mezi komparátory (*NextBottom*, *NextTop*), mezi kterými tyto objekty putují. Na Obrázku 10 lze vidět seznam funkcí a metod této třídy.



Obrázek 10: Seznam vlastností a metod třídy *TestnumberToDraw*

Zbylé třídy, které používám rozšiřují třídu Window obsahující základní třídy pro práci s okny, inicializují veškeré komponenty, aby bylo možné pracovat v okenní aplikaci.

Seznam zbylých tříd

(a) **NewProjectWindow** — okno pro zadání počtu vodičů(Wire), které bude třídící síť obsahovat.

(b) **OwnTestingValuesWindow** — okno pro specifikaci hodnot, pro které chce uživatel síť otestovat.

(c) **SimulationControlWindow** — okno umožňující ovládat průběh simulace(play, pause, skip, stop).

(d) **TestResultsWindow** — okno, které zobrazí výsledky testů.

(e) **TestSettingWindow** — okno pro nastavení komplexnosti simulace.

(f) **WorkingCanvasWindow** — hlavní okno umožňující veškerou práci s programem.

4 Implementace

V této kapitole popisuji, jak je implementováno testování korektnosti třídící sítě, jak načítání třídící sítě skriptem a také popisuji syntaxi mnou navrženého skriptovacího jazyka. Na závěr popíši, jak je implementováno vytvoření a následné spuštění animace.

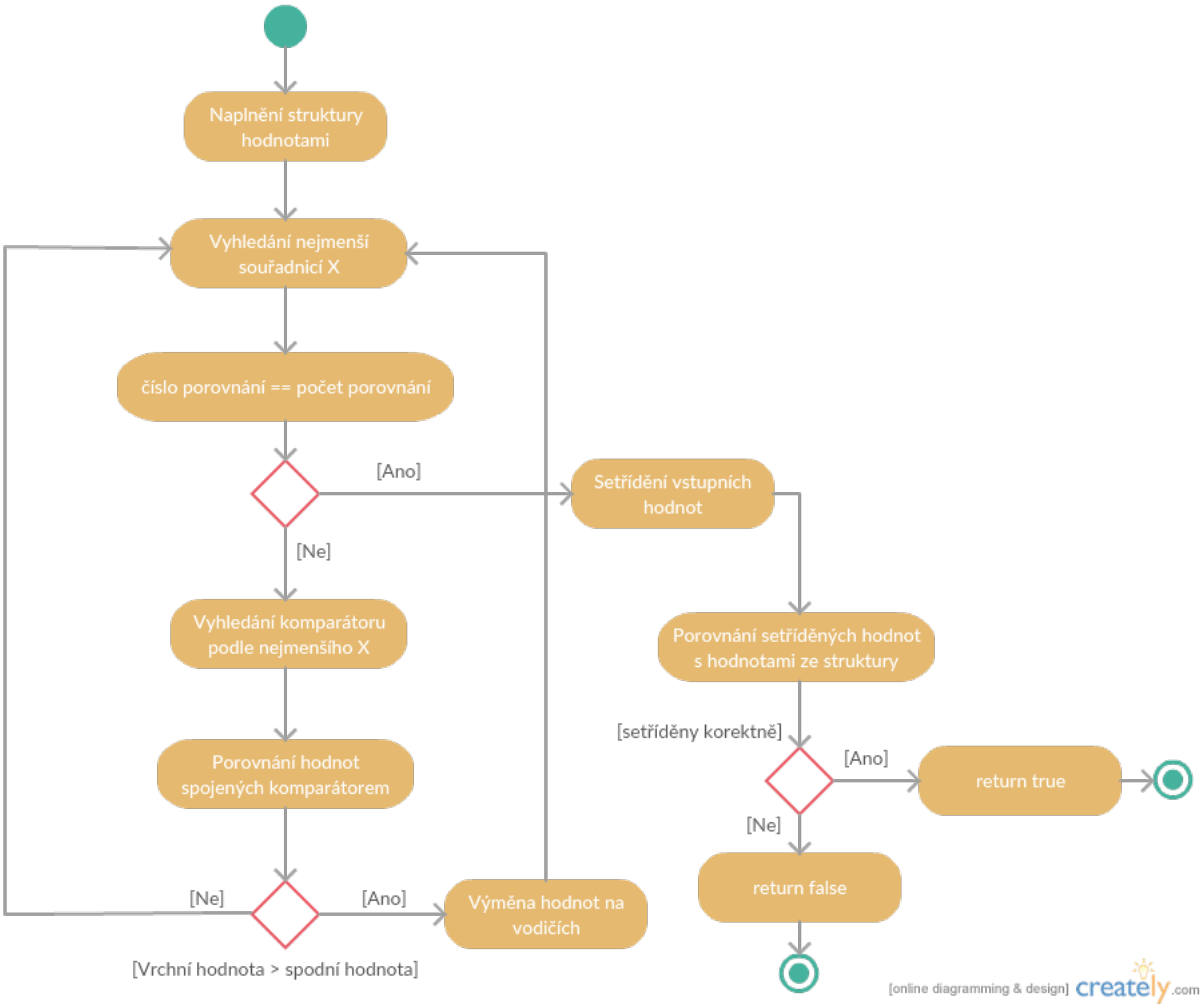
4.1 Testování korektnosti sítě

Pro testování korektnosti třídící sítě (všechny hodnoty jsou setříděny do vzestupného pořadí) se používá veřejná metoda *TestNetwork*. Její první dva vstupní parametry slouží pro vytvoření animace, první parametr je reference na kolekci, do které se budou ukládat objekty, které se následně vykreslují na pracovní plochu (*TestNumberToDraw*), druhý parametr je boolovské hodnoty a určuje jestli se toto uložení provede. Poslední parametr slouží pro naplnění vnitřní struktury vlastními hodnotami, které se zadávají při nastavování testování. Tato metoda má jedno přetížení, kdy v druhé možnosti není potřeba zadávat referenci na pole s vlastními hodnotami k otestování.Samotné testování probíhá ve třech fázích.

1. **Připravení struktury** — v této fázi se do struktury třídící sítě vloží hodnoty, které se budou následně třídit.

2. **Porovnávání hodnot** — zde nastává nejdůležitější část samotného testování korektnosti třídící sítě. Vyhledá se nejmenší souřadnice X v struktuře třídící sítě. Následně se porovná, jestli číslo porovnání je rovno celkovému počtu porovnání. Pokud ne, vyhledá se komparátor ve struktuře s danou souřadnici *X*. Po nalezení toho komparátoru se provede porovnání hodnot, když je vrchní hodnota větší než spodní, tak se tyto dvě hodnoty vymění. Poté se opět vyhledává nejmenší souřadnice a postup se opakuje. Když vrchní číslo nebylo větší, výměna hodnot se neprovádí, ale hned se začíná hledat další nejnižší souřadnice *X*. Tyto operace se provádějí, dokud číslo porovnání není rovno počtu porovnání. Poté, co nastane tato situace, že se tyto dvě čísla rovnají, se tento cyklus ukončí. Po ukončení toho cyklu by měly být hodnoty setříděny do vzestupného pořadí. Nastává třetí fáze testování.

3. **Porovnání posloupností** — hodnoty ze vstupního pole se setřídí do vzestupného pořadí. Následně se tyto dvě vzestupné posloupnosti porovnají, pokud jsou tyto posloupnosti stejné, tak je třídící síť je navrhnuta korektně pro hodnoty, které dostala na vstupu a funkce vrací pravda (true). Když však tyto posloupnosti nejsou shodné, tak se nejedná o třídící síť a funkce vrací nepravda (false).



Obrázek 11: Diagram aktivit pro testování korektnosti sítě

Celá druhá fáze testování je zde implementována za pomocí rekurzivní funkce. Celé testování proběhne vícekrát v závislosti na komplexnosti testu. Průběh testu je popsán pomoci diagramu aktivit, který lze vidět na Obrázku 11.

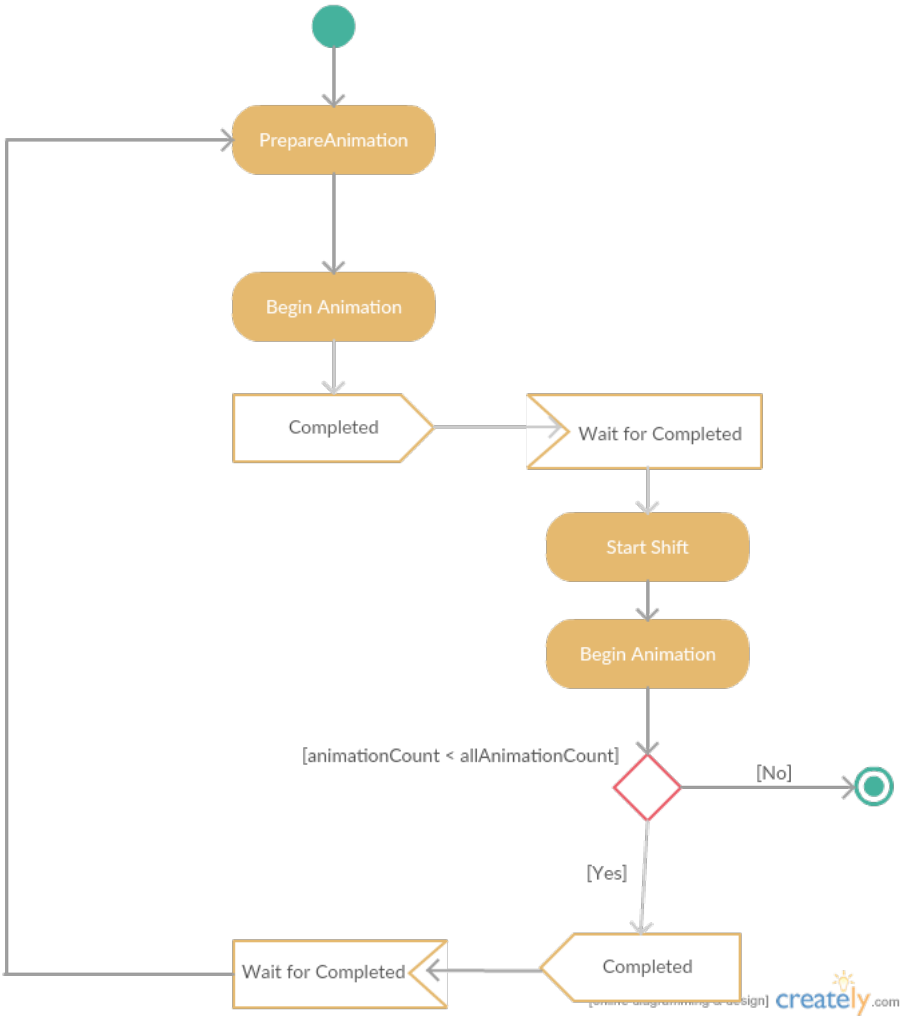
4.2 Vytvoření a spuštění animace

Při testování korektnosti třídící sítě lze vizuálně simulovat, jak se hodnoty třídí. Pro vytvoření a spuštění animace je vytvořen vlastní objekt *TestAnimation*, který obstarává veškerou práci spjatou s touto animaci. Tento objekt má v programu vždy maximálně jednu instanci. Po vytvoření instance se musí zavolat metoda *PrepareStoryboard*.

1. **PrepareStoryboard** - tato metoda po zavolání pracuje s již předdefinovanou kolekcí objektů (*TestNumberToDraw*), ze které vybírá příslušné objekty podle nejnižší souřadnice X pro vykreslení na pracovní plochu za pomocí objektu *Storyboard*. Na konci přípravy první části animace tato funkce zaregistruje událost (event) *StartShift*. Poté připravenou animaci spustí. Proběhne animace výměny dvou čísel, pokud je splněná podmínka, že vrchní hodnota je větší než hodnota spodní.

2. **StartShift** - metoda, která čeká na signál od první animované části. Tento signál obdrží ihned potom, co je první část animace dokončena (Completed). Po přijetí toho signálu o dokončení začne připravovat objekty (*TestNumberToDraw*) z předešlého průběhu pro objekt *Storyboard*. Když jsou tyto objekty připraveny pro animaci, tato metoda zaregistruje událost (event) *ShiftCompleted*, ale pouze v případě, jestli pořadí (index v kolekci) připravovaných objektů je menší než celkový počet objektů. Poté animaci opět spustí. Objekty, které byly touto metodou připraveny, provedou pohyb doprava na předem určenou souřadnici X.

3. **ShiftCompleted** - metoda, která opět čeká na signál o dokončení od předešlé metody *StartShift*. Když tento signál příjme zavolá opět metodu *PrepareStoryboard* a celá série událostí začíná od začátku. Pokud však signál od předešlé metody o dokončení nedostane, tak celá animace je kompletně provedena a cyklus končí.
   Na Obrázku 12 lze vidět tento cyklus zakreslený pomocí diagramu aktivit.



Obrázek 12: Diagram aktivit průběhu animace

Jak lze vidět, celý průběh animace je řízen událostmi, kdy každá událost vždy ovlivní následující průběh celé animace.

4.3 Načítání třídící sítě skriptem

Při načítání třídící sítě z uživatelského skriptu je použit objekt *ScripReader*, který pro načtení skriptu z textového souboru používá metodu *Read*. Tuto metodu v programu vyvolá samotný uživatel po klepnutí na tlačítko a následném vybrání souboru, ze kterého se bude třídící síť načítat.

Seznam všech stavů

Stručný popis, co všechny stavy dělají nebo co očekávají jako vstupní znaky pro korektní načtení třídící sítě z uživatelského skriptu.

1. **INITNETWORK** — očekává na vstupu sekvenci znaků NETWORK.
2. **INITWIRES** — očekává na vstupu sekvenci znaků WIRES.
3. **INITCOMPARATORS** — očekává na vstupu sekvenci znaků COMPARATOR.
4. **BRACE** — očekává na vstupu znak otevřené složené závorky „{ “.
5. **ENDBRACE** — očekává na vstupu znak uzavřené složené závorky „}“.
6. **BRACKET** — očekává na vstupu znak otevřené závorky „(“.
7. **ENDBRACKET** — očekává na vstupu znak uzavřené závorky „)“.
8. **NUMBER** — očekává na vstupu sekvenci číslic.
9. **COMMENT** — po nalezení znaku # ignoruje všechny znaky následující za ním.
10. **NUMBERORCOMMA** — rozhoduje o následujícím stavu podle příchozího znaku čárky nebo číslice.
11. **COMMA** - očekává na vstupu znak čárky.
12. **NEXTCOMPARATOR** - podle příchozího znaku rozhoduje, jestli se budou inicializovat další komparátory nebo nastane stav *ENDBRACE*.
13. **ERROR** — stav, který nastane, pokud při načítání byla nalezena chyba, poté program vyhodí výjimku *BadFormatException*

Příklad takto načteného skriptu lze vidět na Výpisu 1

```
NETWORK
{
   WIRES(5)
   {
      COMPARATOR(0,1,36)
      COMPARATOR(1,5,12)
   }
}
```

Výpis 1: Příklad načítané sítě ze souboru

Read

Metoda, která při načítání třídící sítě používá vlastní vnitřní stavy, které určují jaký znak nebo řetězec očekávají a podle toho, třídící síť vytváří nebo vyhodí vlastní výjimku *BadFormatException*, ze které se uživatel dozví, kde v jeho skriptu nastala chyba. Celkem je těchto stavů třináct. Po zavolání této metody se spustí cyklus, ve kterém se toto načítání zprostředkovává. Celý skript se prochází sekvenčně, takže očekávané vstupy musejí být ve správném pořadí.

První stav je ***INITNETWORK*** v tomhle stavu je na řetězcovém (string) vstupu očekáváno klíčové slovo NETWORK, když je dané klíčové slovo nalezeno, tak program může pokračovat dále a stav se mění na ***BRACE***. Zde program očekává jako vstupní znak otevřenou složenou závorku „{ "pokud cyklus na tento znak narazí, načítání třídící sítě může pokračovat dále.

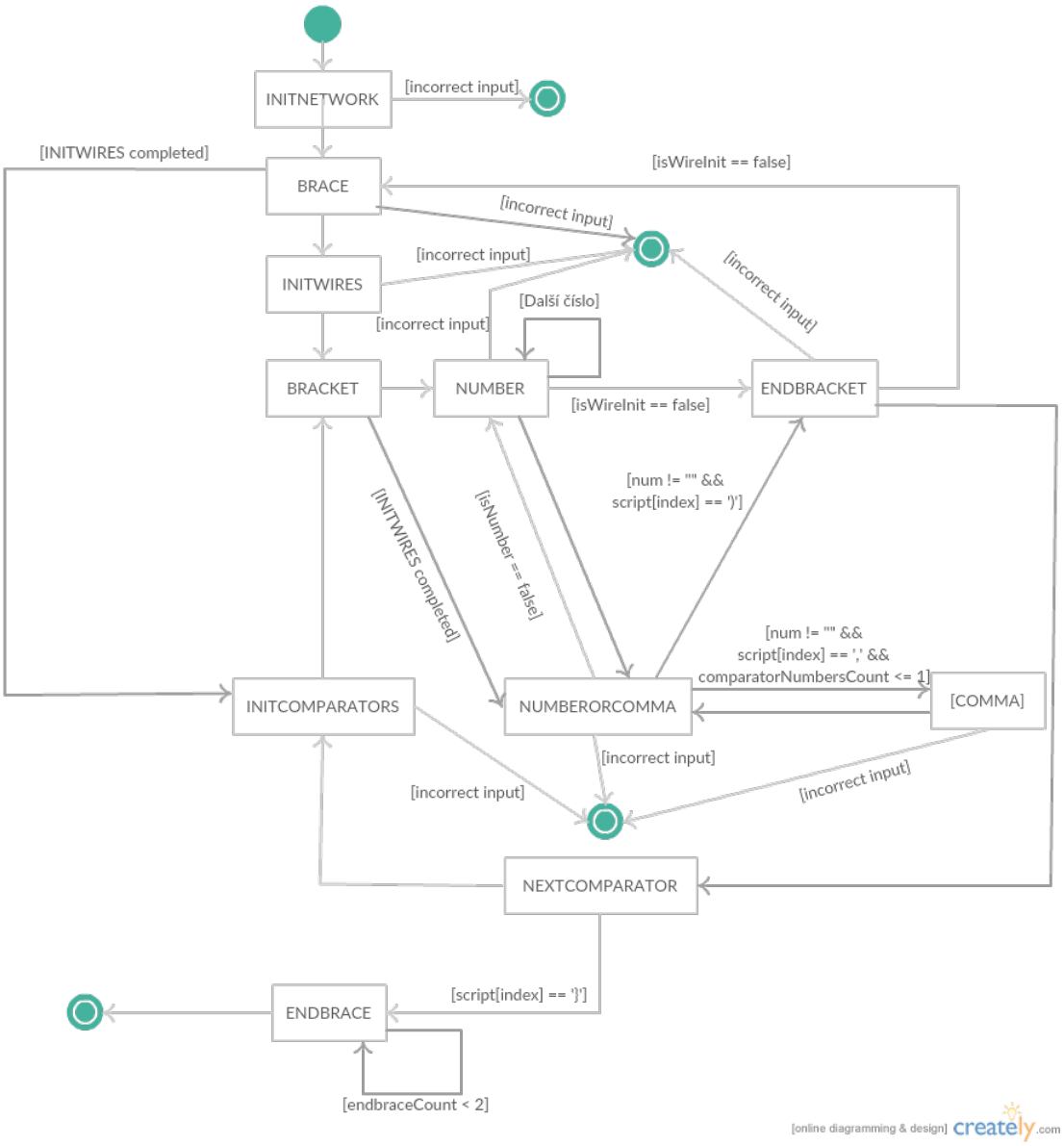
Nastaví se stav ***INITWIRES***. Tento stav očekává na svém vstupu klíčové slovo (posloupnost znaku) „WIRES", pokud je tato posloupnost nalezena, program může dále pokračovat a přistupuje ke stavu ***BRACKET***. Aktivací tohoto stavu program očekává na vstupu znak otevřené závorky „(". Pokud je tento znak nalezen, program aktivuje stav ***NUMBER***. Zde program očekává sekvenci znaků reprezentující číslice, podle které vytvoří daný počet instanci třídy *Wire* nebo v dalších krocích parametrizuje pozici komparátoru. Jestliže program dostane korektní sekvenci číslic, může se přepnout do následujícího stavu ***ENDBRACKET***.

Tento stav je „ukončením"stavu *BRACKET*, to znamená, že program očekává na vstupu uzavřenou závorku„)", která uzavře předchozí závorku a tím ukončí načítání zadaného objektu. Pokud byl tento znak nalezen program přejde do dalšího stavu ***BRACE***, který je popsán již výše, s tím rozdílem, že zde již neočekává program na vstupu podmínky pro *INTWIRES*, ale ***INITCOMPARATORS***. V tomto stavu dochází k samotnému inicializování komparátorů pro porovnávání. Na vstupu je očekávána sekvence znaků COMPARATOR, když program tuto sekvenci znaku nalezne, přepne stav na ***BRACKET***, avšak s tím rozdílem, že následující stav po úspěšném nalezení znaku již není *NUMBER*, ale ***NUMBERORCOMMA***. Zde se rozhodne podle příznaku *number*, jestli na vstup musí přijít číslo nebo nemusí. Pokud musí přijít na vstup číslo, přepne se stav na *NUMBER*, pokud však číslice přijít nemusí, tak očekává na vstup v prvních dvou krocích znak čárky. Když tento znak dostane na vstup, přepne se program do stavu ***COMMA***. Zde poté tento znak zpracuje a přepíná se zpět do stavu *NUMBERORCOMMA*. Tento postup se provádí opět znovu. Ve třetí iteraci již však program neočekává čárku, ale uzavřenou závorku, to znamená, že se přepne do stavu *ENDBRACKET*. Jelikož již proběhl stav *INITWIRES*, tak následující stav bude ***NEXTCOMPARATOR***.

Tento stav rozhoduje podle následujícího znaku, jaký stav bude další. Pokud přijde na řadu jiný znak než znak složené závorky „}", přejde se na stav INITCOMPARATORS a postup se opakuje znovu od tohoto stavu. Jinak se přistoupí ke stavu ***ENDBRACE***. Tento stav očekává jako vstupní znak uzavřenou složenou závorku „}", musí proběhnout dvakrát, aby došlo ke korektnímu uzavření obou složených závorek a tím se načítání třídící sítě korektně ukončilo.

V textu jsem popsal, co nastane pokud je podmínka splněna, když se podmínka nesplní

(nepřejde se k dalšímu stavu) program vyhodí výjimku (*BadFormatException*), která je v programu dále zpracována a slouží k informování uživatele o chybách v jeho skriptu. Dále také je programově ošetřeno načítání bílých znaků (mezera, tabelátor, enter), které se v sekvenci znaků ignorují. Dále může nastat stav ***COMMENT*** a to tehdy, když v sekvenci znaků program nalezne znak # poté se ignoruje sekvence znaků za ním na daného řádku, na kterém se tento znak nachází. Po úspěšném průběhu tohoto cyklu se vytvořená síť vykreslí do pracovního okna. Celý průběh načítání je popsán za pomoci stavového diagramu, který lze vidět na Obrázku 13.



Obrázek 13: Stavový diagram pro načtení třídící sítě pomoci skriptu.

5 Návod pro uživatele

V této kapitole popíší, co vše je třeba mít pro správné spuštění programu. Následně popíši veškeré možnosti, které program nabízí uživateli pro navrhování třídících sítí. Na závěr uvedu také možnosti, jak vytvářet třídící síť pomocí skriptovacího jazyka, který byl navržen pro tento program. Rozeberu syntaxi a uvedu příklad sítě.

5.1 Požadavky a instalace programu

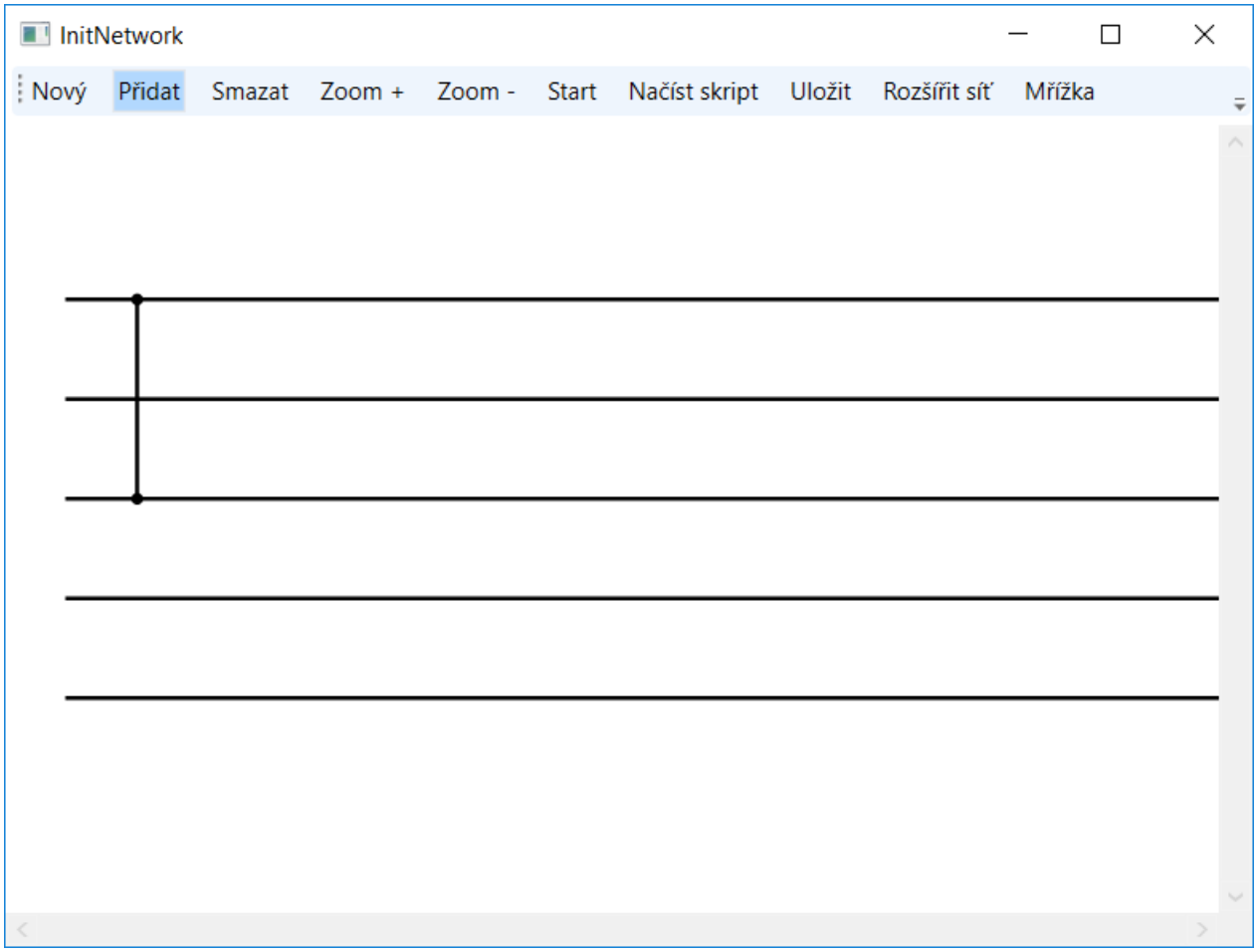
Instalace programu je velmi jednoduchá, avšak jsou zde specifické požadavky. Program byl vyvinut na .NET frameworku na platformě windows. Program by měl bez problému fungovat na .NET frameworku ve verzi 4.0 na windows platformě od verze windows 7, avšak vývoj a testování bylo prováděno na windows 10 s .NET frameworkem ve verzi 4.5. Pro samotnou instalaci programu stačí pouze z CD zkopírovat soubor SortingNetwork.exe kdekoliv do svého počítače a potom jej spustit.

5.2 Práce v grafickém editoru

Jak již bylo výše zmíněno, v grafickém editoru si může uživatel navrhovat interaktivně třídící sítě dle svého uvážení a dále pak může testovat korektnost jejich navržení.

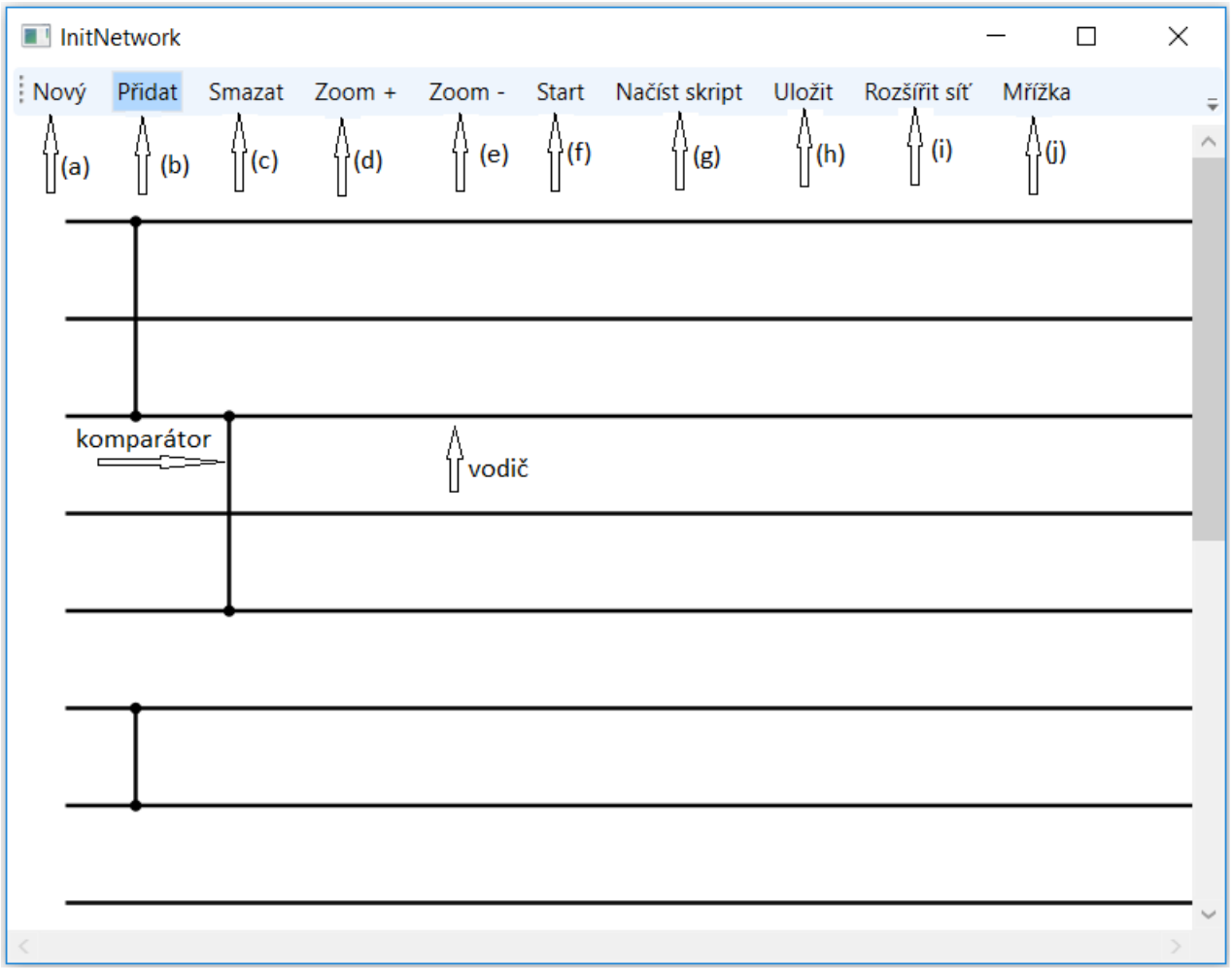
Práce s programem

Na začátek je nutné vytvořit nový projekt nebo načíst již rozpracovaný projekt, který se ukládá v podobě skriptu. Po klepnutí na tlačítko „Nový“ se otevře nové okno pro založení projektu, kde je nutno zadat počet vodičů, které tato síť bude obsahovat. Po klepnutí na tlačítko vytvořit, se vygeneruje zadaný počet vodičů, se kterými lze následně pracovat. Nyní již lze přidávat do třídící sítě komparátory. Pro možnost přidání komparátoru do třídící sítě je nutno mít zakliknuté tlačítko „Přidat“, které toto přidávání umožňuje. Komparátor se nakreslí jednoduchým klepnutím na dva různé vodiče a tím se přidá komparátor do struktury třídící sítě.Po provedení následujících kroků by mohl program vypadat jako na Obrázku 14.



Obrázek 14: Mód pro vytvoření komparátoru

Takto se postupuje při vytváření sítě stále dokola. Pokud je komparátor vytvořen na nesprávném místě může být buď přemístěn nebo odstraněn. Přemístění komparátoru se zahájí, pokud se klikne na daný komparátor, který bude přemisťován a pohybem myši se přesouvá na správnou pozici. Toto přemístění lze provádět pouze v horizontálním směru. Špatně nakreslený komparátor lze však odstranit a to zakliknutím tlačítka smazat. Tím se změní i kurzor myši a poté už jen stačí kliknout na špatně vytvořený komparátor. V programu lze pracovní plochu různě přibližovat a oddalovat po klepnutí na tlačítko „Zoom+“ provede přiblížení nebo „Zoom-“, které provede oddálení pracovní plochy. Třídící síť, která je nakreslená na pracovní ploše programu, lze uložit po klepnutí na tlačítko uložit. Poté se jen zadá název souboru a potvrdí se. Tím se vytvoří textový soubor, který obsahuje skript, který slouží k opětovnému načtení třídící sítě do programu. Při navrhování se může stát, že délka jednotlivých vodičů nedostačuje. Po kliknutí na tlačítko rozšířit síť se délka jednotlivých vodičů prodlouží. Na Obrázku 15 lze vidět popis jednotlivých funkcí, pod ním jsou popsány funkce jednotlivých tlačítek.



Obrázek 15: Popis jednotlivých funkcí programu

Funkce jednotlivých tlačítek

Popis tlačítek se vztahuje k Obrázku 15

(a) **Nový** — otevře nové okno, kde uživatel zadá počet vodičů, se kterým bude následně pracovat.

(b) **Přidat** — program je přepnut do módu pro kreslení jednotlivých komparátorů.

(c) **Smazat** — program je přepnut do módu, kde po klepnutí na jednotlivé komparátory budou tyto smazány.

(d) **Zoom+** — přiblíží pracovní plochu.

(e) **Zoom-** — oddálí pracovní plochu.

(f) **Start** — otevře nabídku pro testování navrhnuté třídící sítě.

(g) **Načíst skript** - otevře dialogové okno pro vybrání souboru, ve kterém se nachází skript pro vytvoření třídící sítě.

(h) **Uložit** — otevře dialogové okno pro uložení uživatelem nakreslené sítě.

(i) **Rozšířit síť** — prodlouží všechny vodiče na pracovní ploše.

(j) **Mřížka** — zobrazí na pracovní ploše mřížku, ke které se komparátory zarovnávají.

5.3 Vytvoření sítě pomocí skriptu

Třídící síť si uživatel může také vytvořit pomocí skriptu. Jazyk byl vytvořen speciálně pro tento program a proto obsahuje jednoduché konstrukce zápisu sítě. Používá tři klíčová slova *NETWORK*, *WIRES*, *COMPARATOR*.

Klíčové slovo NETWORK inicializuje celou třídící síť. Následně je nutno zadat, kolik bude mít třídící síť vodičů, kterými budou probíhat hodnoty k porovnání. K tomu slouží klíčové slovo WIRES a jako parametr číslo <wires count>, jež určí kolik vodičů se vytvoří. Následně již stačí zadávat jednotlivé komparátory. Pro vytvoření komparátoru se používá klíčové slovo COMPARATOR, kde je nutno dosadit parametry, <wire 1>, <wire 2> číslo prvního a druhého vodiče (vodiče jsou číslovány od nuly), ke kterým je komparátor připojen a jako poslední parametr <coordinates X>, který určí, kde na vodiči se daný komparátor bude nacházet. Při tvorbě sítí lze taky používat v souboru komentáře pro jakékoliv uživatelské poznámky. Obecný popis tohoto lze vidět na Výpisu 2.

```
NETWORK
{
   WIRES(<wires count>)
   {
      COMPARATOR(<wire 1>, <wire 2>, <coordinates X>)
      # This is comment.
      ...
   }
}
```

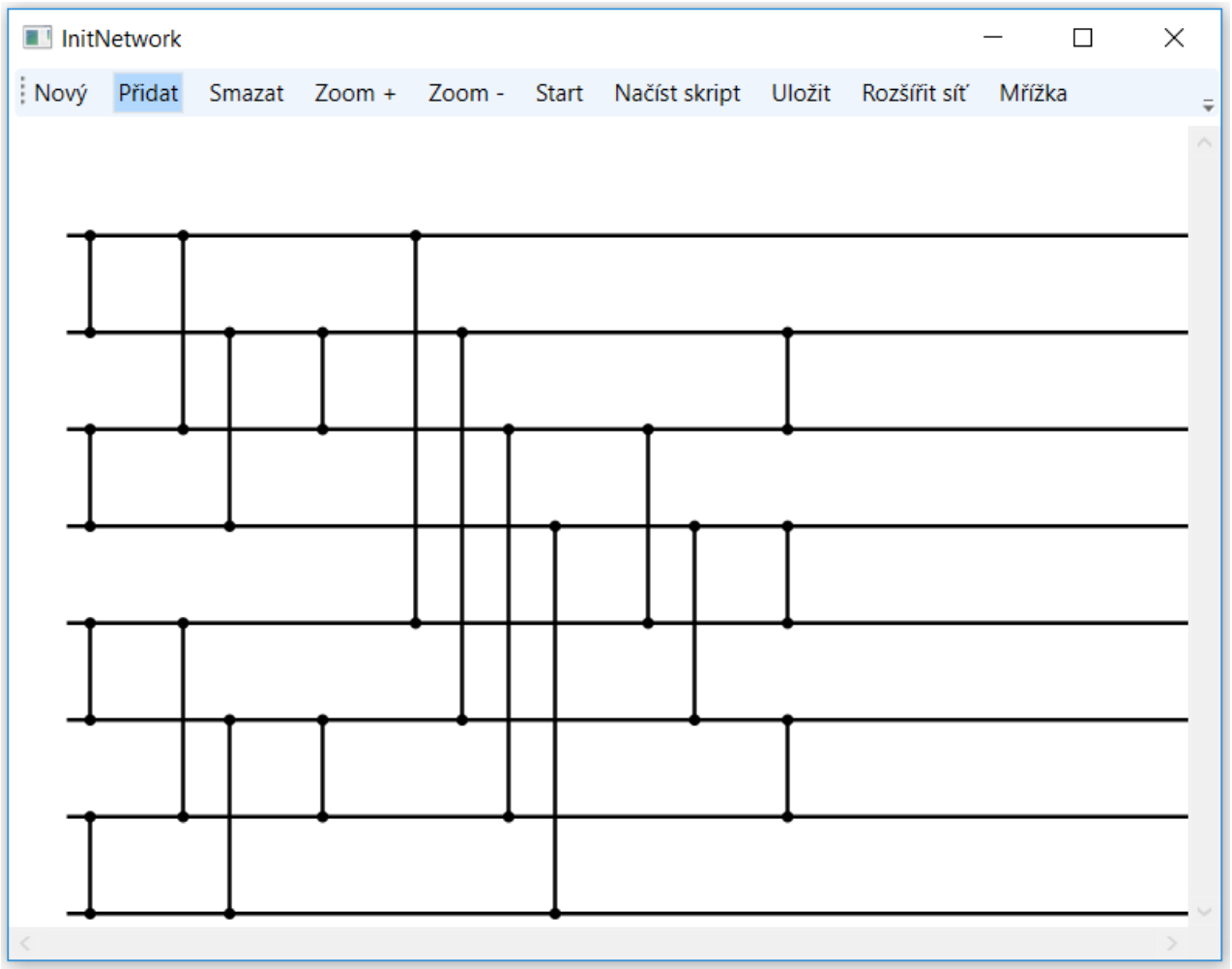
Výpis 2: Obecný zápis pro vytvoření třídící sítě

Jelikož vytváření třídících sítí za pomocí skriptu není nijak omezováno, je dobré dodržovat odstup mezi komparátory na stejném vodiči alespoň 24 bodů, protože podle této hodnoty se pozice komparátorů přepočítává, aby třídící síť zůstala přehledná a její testování probíhalo korektně. Pro příklad, pokud by souřadnice X měla hodnotu 31, tak po přepočtu by souřadnice měla hodnotu 36. Příklad korektního skriptu pro vytvoření třídící sítě lze vidět na Výpisu 3.

```
NETWORK{
   WIRES(8)
   {
      COMPARATOR(0,1,12)
      COMPARATOR(0,2,60)
      COMPARATOR(0,4,180)
      COMPARATOR(1,3,84)
      COMPARATOR(1,2,132)
      COMPARATOR(1,5,204)
      COMPARATOR(1,2,372)
      COMPARATOR(2,3,12)
      COMPARATOR(2,6,228)
      COMPARATOR(2,4,300)
      COMPARATOR(3,7,252)
      COMPARATOR(3,5,324)
      COMPARATOR(3,4,372)
      COMPARATOR(4,5,12)
      COMPARATOR(4,6,60)
      COMPARATOR(5,7,84)
      COMPARATOR(5,6,132)
      COMPARATOR(5,6,372)
      COMPARATOR(6,7,12)
   }
}
```

Výpis 3: Příklad skriptu pro algoritmus Merge sort.

Z kódu lze vyčíst, že třídící síť bude zkonstruována na pěti vodičích a bude mít velikost (počet komparátorů) 10. Jak by po načtení tohoto skriptu programem vypadala tato třídící síť lze vidět na Obrázku 16. Tento algoritmus je popsán v Kapitole 2 a lze vidět na Obrázku 3b.



Obrázek 16: Ukázka programu po načtení výše vypsaného skriptu.

Na CD, které je přiloženo k této bakalářské práci je ještě mnoho dalších předpřipravených algoritmů k vyzkoušení.

6 Závěr

Výsledkem této bakalářské práce je program pro práci s třídícími sítěmi. Tento program umožňuje interaktivní vytváření, upravování třídících sítí v grafickém prostředí aplikace a následné testování jejich korektnosti. V neposlední řadě umožňuje vizualizaci průběhu tříděných hodnot, což může uživateli usnadnit odhalování chybně umístěných komparátorů při navrhování těchto třídících sítí.

Program dále zvládá programové načítání třídících sítí, které uživatel může vytvářet v textovém editoru (například poznámkový blok), při použití syntaxe, která byla speciálně navržena pro popis třídících sítí v této bakalářské práci. Program také dokáže uložit do textové podoby uživatelem navrženou třídící síť v interaktivním prostředí a to umožní uživateli kdykoliv si tuto síť znovu načíst a pokračovat v jejím návrhu a testování.

K úspěšné implementaci tohoto programu bylo zapotřebí nastudovat problematiku třídících sítí. Tyto znalosti dopomohly k vhodnému navržení uživatelského prostředí pro tento program.

Celý program je tedy realizován jako desktopová aplikace fungující na platformě windows.

Literatura

[1] Sorting network. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2017 [cit. 2017-04-18].
Dostupné z: `https://en.wikipedia.org/wiki/Sorting_network`

[2] Bubble sort. Algoritmy [online]. Česká Republika: Jan Neckář, 2016 [cit. 2017-04-18].
Dostupné z: `https://www.algoritmy.net/article/3/Bubble-sort`

[3] Merge sort. Algoritmy [online]. Česká Republika: Jan Neckář, 2016 [cit. 2017-04-18].
Dostupné z: `https://www.algoritmy.net/article/13/Merge-sort`